Dylan Park and Kira Lauring’s Design Document

PART A

1. *What is the abstract thing you are trying to represent?*

We are trying to represent an unboxed two-dimensional array that can hold any type of data. We’ll call it a *UArray2*.

2. *What functions will you offer, and what are the contracts of that those functions must meet?*
    a. `UArray2_map_row_major(T uarray2, void apply(int row, int col, void *elem, void *cl), void *cl)` calls apply function for each element in the array, row by row
    b. `UArray2_map_col_major(T uarray2, void apply(int row, int col, void *elem, void *cl), void *cl)` calls apply function for each element in the array, column by column
    c. `extern T UArray2_new(int height, int width, int size)` creates a new 2-D array of the specified height and width, preparing it to hold elements of size bytes and returns an initialized UArray2
    d. `extern void UArray2_free(T *uarray2)` frees all the memory allocated within 2-D array
    e. `extern int UArray2_height(T uarray2)` returns the height of the 2-D array
    f. `extern int UArray2_width(T uarray2)` returns the width of the 2-D array
    g. `extern int UArray2_size(T uarray2)` returns the number of bytes found in one element of the 2-D array
    h. `extern void *UArray2_at(T uarray2, int row, int col)` returns address of the element found at (i,j)

3. *What examples do you have of what the functions are supposed to do?*

```
UArray2_T UArray2 = UArray2_new(5, 7, 1); // creates new 2-D array
UArray2_height; // returns 5
UArray2_width; // returns 7
UArray2_size; // returns 1
index_p = UArray2_at(Uarray2, 4, 5); // returns address of (4,5)
Elem_type *index = index_p; // sets pointer to element type to address to
which index_p points
*index = Elem; // add element to this place in the array
UArray2_map_row_major(UArray2, print, NULL); // prints every element of
the array by row first and column second
UArray2_free(Uarray2); // should free all memory allocated for both
instances of UArray
```

Other example use case: If functions are called on a null Uarray2, they will raise an exception.

4. *What representation will you use, and what invariants will it satisfy?*

We will be using a UArray with a length equivalent to the number of columns. Each “index” of this UArray will hold another UArray with a length equal to the number of rows. The elements themselves are stored in these secondary UArrays. Together, this will represent a 2-D array. The invariants our code will satisfy include:

- UArray must contain at least one element.
- Element number i is stored at address Uarray(col array)->elems + i*Uarray(col array)->size.
- UArray's elements must all be the same size
- Row and col passed in as arguments for `Uarray2_at` and `Uarray2_new` must be greater than 0 and must be less than or equal to height and width, respectively.

5. *How does an object in your representation correspond to an object in the world of ideas?*

We create one UArray with as many indices as there are columns. Within this UArray, each index holds a UArray of its own which represents the columns themselves. The culmination of first index of each column represents the entire first row. This trend continues for the second index of each of the columns, the third, etc.

6. *What test cases have you devised?*
   - Trying to access memory that is out of bounds
     - Calling `UArray2_at(uarray2, 6, 6)` on the same 5x5 array should raise an exception for an out of bounds reference.
   - Trying to access memory that is out of bounds for one parameter and in-bounds for another
     - Calling `UArray2_at(uarray2, 2, 6)` on the same 5x5 array should raise an exception for an out of bounds reference.
   - Calling `UArray_new` with an incorrect `size` value
     - This should raise an exception due to incorrect allocation of memory.
   - Adding elements of different sizes, such as a int and a double, to the same Uarray2
     - This should raise an exception due to incorrect allocation of memory.
   - Testing Uarray2_at for elements at different coordinates within the bounds of the array
     - Corners such as (1, 1) and (height, width)
     - Same row and column
     - Different row and column
     - Filling entire array
     - For all the cases above, *(type *) Uarray2_at(array, row, col) will be the element at that coordinate.
   - Calling all Uarray2.h functions, passing them a null array
     - Should raise an exception

7. *What programming idioms will you need?*
   - The idiom for getting a pointer to the i-th element
   - The idiom for storing values into an unboxed array
   - The idiom for making an array of arrays
   - The idiom for correctly allocating memory
   - The idiom for handling void pointer values of known type

- The idiom for type abbreviations for structure types

PART B

1. *What is the abstract thing you are trying to represent?*

We are trying to represent an unboxed two-dimensional array that can holds bits. We'll call it a *Bit2.*

2. *What functions will you offer, and what are the contracts of that those functions must meet?*

   `Bit2_map_row_major(T bit2, void apply, void *cl)` calls apply function for each element in the array, row by row
   `Bit2_map_col_major(T bit2, void apply, void *cl)`calls apply function for each element in the array, column by column
   `extern T Bit2_new(int height, int width, int size)` creates a new 2-D array of the specified height and width
   `extern void Bit2_free(T *bit2)` frees all the memory allocated within 2-D array
   `extern int Bit2_height(T bit2)` returns the height of the 2-D array
   `extern int Bit2_width(T bit2)` returns the width of the 2-D array
   `extern int Bit2_size(T bit2)` returns the number of bytes found in one element of the 2-D array
   `extern int Bit2_put(T bit2, int row, int col, int bit)` changes the bit at (row, col) to value of bit and returns the previous value of bit
   `extern int Bit2_get(T bit2, int row, int col)`returns the bit found at (row, col)

3. *What examples do you have of what the functions are supposed to do?*

```
Bit2_T Bit2 = Bit2_new(5, 7, 1); // creates new 2-D array of bits (calls
UArray_new within function)
Bit2_height; // returns 5
Bit2_width; // returns 7
Bit2_size; // returns 1
Bit2_put(Bit2, 2, 3, 1); // set bit at (2, 3) to 1
printf("%i", Bit2_get(Bit2, 2, 3)); //prints bit at (2, 3)
Bit2_map_row_major(Bit2, print, cl); // prints every bit in the array by
row first and column second
Bit2_free(Bit2); // should free all memory allocated for both instances
of Bit where Bit 2 is a pointer
```

4. *What representation will you use, and what invariants will it satisfy?*

We will be using a UArray with a length equivalent to the number of columns. Each element of this UArray will hold a bit vector with a length equal to the number of rows. The bits themselves are stored in this bit vector. The invariants of our code will be:

- Bit2 must have at least one element
- All elements must be bits
- Each bit must be 0 or 1
- Row and col passed in as arguments for put, get, and new must be positive

5. *How does an object in your representation correspond to an object in the world of ideas?*

We create one UArray with as many indices as there are columns. Within this UArray, each index holds a bit vector of its own which represents the columns themselves. The first index of each of the different bit vectors represents the entire first row. This trend continues for the second index of each of the bit vectors, the third, etc.

6. *What test cases have you devised?*
    - Trying to access memory that is out of bounds
        - Passing row and col arguments that are negative or greater than the height/with on Bit2_put(), Bit2_get(), or Bit2_new() should raise an exception for an out of bounds reference.
    - Calling `Bit2_new` with an incorrect `size` value
        - This should raise an exception due to incorrect allocation of memory.
    - Putting non-bits into Bit2
        - Should throw an exception
    - Putting in elements of different sizes, such as a int and a double, to the same Bit2
        - This should raise an exception due to incorrect allocation of memory.
    - Testing `Bit2_get` for elements at different coordinates within the bounds of the array
        - Corners such as (1, 1) and (height, width)
        - Same row and column
        - Different row and column
        - Calling for all elements of a full array
        - For all the cases above, *(type *) Bit2_get(bit_vector, row, col) will be the element at that coordinate.
    - Calling all Bit2.h functions with a null array
        - Should raise an exception

7. *What programming idioms will you need?*
    - The idiom for storing values into an unboxed array
    - The idiom for making an array of arrays (but it will be a Uarray of bit vectors so slightly modified but same idea)
    - The idiom for correctly allocating memory
    - The idiom for handling void pointer values of known type
    - The idiom for type abbreviations for structure types